Final Report: Knee MRI Dataset Analysis Using SVM and CNN

YeonJun Lee, Omar Badr, Vasista Dhyasani

December 13, 2024

Abstract

This report presents an analysis of the Knee MRI dataset using Support Vector Machines (SVM) and Convolutional Neural Networks (CNN). The dataset consists of MRI scans processed and evaluated for binary classification. The primary focus of this study was to preprocess the data, build predictive models using SVM and CNN, and evaluate their performance. The results include accuracy metrics, a discussion of class imbalance issues, and a comparison of the benefits and drawbacks of each approach.

1 Introduction to the Problem and Dataset

Magnetic Resonance Imaging (MRI) is a non-invasive diagnostic tool widely used in medical applications. Automating the analysis of MRI scans can significantly improve diagnostic efficiency and accuracy. Machine learning techniques, especially Support Vector Machines (SVM) and deep learning approaches like Convolutional Neural Networks (CNN), can be promising in this area.

The Knee MRI dataset used in this study is organized into multiple subfolders containing pickle (.pck) files. Each file represents an MRI scan. Preprocessing was required to standardize the data format and address challenges such as class imbalance.

Dataset: https://www.kaggle.com/datasets/sohaibanwaar1203/kneemridataset

2 Support Vector Machines (SVM)

2.1 SVM Methodology

The SVM model was trained using the Radial Basis Function (RBF) kernel. Data was flattened into vectors to meet the input requirements of the SVM classifier. The model was evaluated using accuracy and classification metrics.

Listing 1: SVM Training and Evaluation

```
from sklearn import svm
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import train_test_split

def train_svm(X_train, y_train):
    # Train the SVM classifier with class weights
    clf = svm.SVC(kernel='rbf', gamma='scale', C=100.0, class_weight={0: 2, 1: 5})
    clf.fit(X_train, y_train)
    return clf

def main():
    ### Additional Code ###

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
    print('* Split the data into train/test')

    # Resample the training set

    under_sampler = RandomUnderSampler(random_state=42)
    X_train_resampled, y_train_resampled = under_sampler.fit_resample(X_train, y_train)
    print('* Resampled the training data to undersample majority class')
```

```
print(y_train)
print(y_train_resampled)

# Train SVM
clf = train_svm(X_train_resampled, y_train_resampled)
print('* Trained the SVM model')

# Make predictions and evaluate
y_pred = clf.predict(X_test)

### Additional Code ###
```

2.2 SVM Results

The model achieved an accuracy of 62.5%, with higher precision for class 0 (healthy ACL) at 81%, but lower precision for class 1 (damaged ACL) at 42%. The recall for class 1 was relatively high at 67%, indicating that the model is fairly good at identifying damaged ACLs, but it struggles with precision, resulting in many false positives. For class 0, the recall was 61%, showing that the model correctly identifies a good portion of healthy ACLs. The F1-scores reflect these issues, with class 0 scoring 0.69 and class 1 scoring 0.52. Given the class imbalance (138 healthy ACLs vs. 62 damaged ACLs), the model performs better at identifying the majority class (healthy ACL), but more work is needed to improve precision for the minority class (damaged ACL).

2.3 Transition to Binary Classification

To address the challenges of multi-class classification, we decided to simplify the problem by merging the two damaged ACL classes (Class 1 and Class 2) into a single "damaged ACL" class. This conversion to a binary classification setup—**Class 0 (Healthy ACL)** and **Class 1 (Damaged ACL)**—allowed us to do two things:

- **Reduce Computational Demands**: Training the model on only two classes was computationally less expensive and more manageable for our machine.
- **Reduce Class Imbalance**: By combining the two minority classes, we reduced the imbalance between healthy and damaged ACLs.

Despite these improvements, we still faced a significant class imbalance, with a large number of healthy ACLs (Class 0) and a smaller number of damaged ACLs (Class 1). This imbalance skewed the model's predictions heavily towards Class 0. As a result, the model performed well at identifying healthy ACLs, but struggled to accurately classify damaged ACLs.

2.4 Addressing Class Imbalance

Class imbalance remained a key challenge. With the majority of the data belonging to Class 0 (healthy ACLs), the SVM model tended to heavily favor this class. This imbalance caused the hyperplane to be skewed towards Class 0, making the model much better at predicting healthy ACLs than detecting damaged ACLs.

To address this issue, we explored two main strategies to mitigate the imbalance:

- **Oversampling the Minority Class**: We initially tried using SMOTE (Synthetic Minority Oversampling Technique) and RandomOverSampler to generate synthetic data points for the minority class (Class 1). However, we found these approaches to be computationally expensive. The process of generating synthetic points required calculating the distance between data points and their neighbors, which increased the computational cost exponentially as the number of synthetic points grew.

- **Undersampling the Majority Class**: To reduce the class imbalance in a more manageable way, we chose to undersample the majority class (Class 0). By removing some of the data points from Class 0, we aimed to create a more balanced training set. While this approach allowed us to reduce the class imbalance, it still didn't fully resolve the issue, and the model continued to overpredict Class 0. However, we decided to go with this approach as opposed to oversampling.

2.5 Hyperparameter Tuning

With the class imbalance still affecting the model's performance, we turned to hyperparameter tuning in an attempt to improve the model's ability to classify damaged ACLs (Class 1). Since using automated tuning methods like GridSearchCV would be too slow and computationally expensive, we opted to fine-tune the hyperparameters manually. Specifically, we adjusted the following:

- **Regularization Parameter (C)**: We set $C = 100.0$, which increased the penalty for misclassification and helped make the model more sensitive to the minority class.

- **Gamma**: We used the 'scale' option for gamma, which adapts the kernel to the data.

- **Kernel**: We used the Radial Basis Function (RBF) kernel, which is suitable for capturing non-linear relationships between data points.

- **Class Weights**: To further address the imbalance, we set class weights as $\{0 : 2, 1 : 5\}$, meaning Class 1 (damaged ACLs) was given 2.5 times more weight than Class 0 (healthy ACLs). This would encourage the SVM to focus more on correctly classifying damaged ACLs.

Despite these adjustments, the model still had challenges in accurately identifying damaged ACLs. The precision for Class 0 was high, indicating the model's strong ability to predict healthy ACLs, but the precision for Class 1 remained relatively low, suggesting that the model struggled with false positives in the damaged ACL class.

2.6 Conclusion and Future Directions

In summary, we encountered several challenges while working with the SVM model:

- **Class Imbalance**: Even after merging the two damaged ACL classes, the imbalance between Class 0 and Class 1 remained a significant challenge, skewing the hyperplane towards the majority class.
- **Computational Cost**: Techniques like SMOTE for oversampling proved too computationally intensive, leading us to opt for undersampling.
- **Hyperparameter Tuning**: While manual tuning of hyperparameters improved the model's performance, further refinement of the model is necessary to improve its ability to classify damaged ACLs accurately.

Given these findings, future work could focus on more advanced techniques for handling class imbalance, such as:

- Using other resampling techniques like ADASYN (Adaptive Synthetic Sampling) for more efficient oversampling.
- Experimenting with ensemble methods, such as Random Forest or XGBoost, which may handle imbalances more effectively.
- Exploring deep learning approaches (like Convolutional Neural Networks) that can automatically learn spatial features from the MRI images, potentially reducing the reliance on manually tuned parameters.

Through these improvements, we hope to achieve a more balanced model that can accurately classify both healthy and damaged ACLs with higher precision and recall.

```
Accuracy: 0.625
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.61      0.69        28
           1       0.42      0.67      0.52        12

    accuracy                           0.62        40
   macro avg       0.62      0.64      0.61        40
weighted avg       0.69      0.62      0.64        40

ACL healthy count (class 0): 138
ACL damage count (class 1): 62
```

3 Convolutional Neural Networks (CNN)

3.1 CNN Methodology

A CNN was built using TensorFlow/Keras for classifying MRI scans. It included convolutional layers to extract features, pooling layers to reduce size, and a dense layer for final predictions. Images were resized and normalized before training. The model was trained over several epochs using the Adam optimizer and cross-entropy loss.

3.2 Transition to CNN-Based Classification

The CNN methodology for our project was developed to classify MRI slices into two categories: healthy ACLs (class 0) and damaged ACLs (class 1). We started by preparing and preprocessing the MRI data. Each MRI volume contained multiple slices, stored in '.pck' files, and we focused on extracting the middle slice from each volume. This was chosen because the middle slice was likely to provide the most relevant information for classification. These slices were resized to 320x320 pixels to ensure a consistent input size for the CNN model. After resizing, we normalized the pixel values to have zero mean and unit variance, which helps the model train more effectively. Labels were derived from filenames, assigning 0 for healthy ACLs and 1 for damaged ACLs.

3.3 Data Splitting and Loading

The data was then split into training and test sets, with 80% used for training and 20% for testing. This allowed us to evaluate the model's performance on unseen data. A PyTorch dataset class was created to handle the images and labels, and data loaders were used to manage batches during training. The data loader also shuffled the training data to ensure better generalization.

3.4 CNN Architecture Design

The CNN model was designed to extract features and classify the MRI slices. It consisted of multiple layers that processed the images step by step. The first part of the model extracted features using convolutional layers, ReLU activation functions, and max-pooling layers. These layers helped the model focus on important patterns in the images, such as edges and textures. As the data passed through these layers, the model gradually increased the number of filters to capture more complex features. At the end of this feature extraction process, an adaptive average pooling layer reduced the spatial size of the data to make it easier for the fully connected layers to process.

The final part of the CNN, the classifier, used fully connected layers to make predictions. It started by reducing the features to 64 dimensions and applied dropout to prevent overfitting. The last layer outputted two values, one for each class, representing the model's confidence in the classification.

The model was trained using a loss function called cross-entropy loss, which measures how well the predictions match the true labels. An optimizer called Adam was used to update the model's weights during training. The learning rate for the optimizer was set to 0.001, which helped the model learn at a

steady pace. The training process ran for 50 epochs, and at the end of every five epochs, we evaluated the model on the test set to check its accuracy.

Class imbalance was a challenge because there were more healthy ACL samples than damaged ones. This imbalance caused the model to favor predicting healthy ACLs. To address this, we adjusted the training process by balancing the samples and experimenting with different ways to weight the loss function. This helped the model pay more attention to the damaged ACL class without losing accuracy for the healthy ACL class.

After training, the model's performance was evaluated using metrics like accuracy, precision, and recall. A classification report and confusion matrix were also generated to understand how well the model distinguished between the two classes. The methodology helped us develop a model capable of identifying damaged ACLs effectively, despite the challenges of limited data and class imbalance.

Listing 2: CNN Training and Evaluation

```
def train_model(model, train_loader, test_loader, criterion, optimizer, num_epochs=50):
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for i, (inputs, labels) in enumerate(train_loader):
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        # Print epoch statistics
        epoch_loss = running_loss / len(train_loader)
        print(f'Epoch [{epoch+1}/{num_epochs}] Loss: {epoch_loss:.4f}')

        # Evaluate every 5 epochs
        if (epoch + 1) % 5 == 0:
            model.eval()
            correct = 0
            total = 0
            with torch.no_grad():
                for inputs, labels in test_loader:
                    inputs, labels = inputs.to(device), labels.to(device)
```

```
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f'Test Accuracy: {accuracy:.2f}%')
    model.train()
```

3.5 CNN Results

The CNN model performed better than the SVM model, reaching an accuracy of 88.51%. This is because CNNs are better at understanding patterns in the data. Making small adjustments to the model and adding more varied training data could help improve the results even further.

3.6 Classification Report

The detailed classification report is presented below:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.49      0.63        69
           1       0.68      0.94      0.79        79

    accuracy                           0.73       148
   macro avg       0.78      0.71      0.71       148
weighted avg       0.77      0.73      0.71       148
```

4 Comparison of SVM and CNN

4.1 Benefits and Drawbacks

SVM is simple and works well with small datasets, but it struggles with large, complex data unless you carefully prepare the features. CNN is better at finding complex patterns in data, but it needs more computing power and larger datasets to work well and avoid mistakes.

4.2 Why CNN Outperformed SVM

Convolutional Neural Networks (CNNs) generally outperformed Support Vector Machines (SVMs) in classifying knee MRIs for several key reasons:

First, CNNs automatically learn relevant features at each layer of the network. Early layers capture simple details like edges and textures, middle layers recognize patterns, and deeper layers identify more

complex structures such as anatomical features. This hierarchical learning process allows CNNs to automatically adapt to the specific needs of the dataset. In contrast, SVMs rely on manually chosen features or raw pixel data, which makes it difficult to capture the intricate patterns and important details present in medical images. The need for extensive feature engineering limits the effectiveness of SVMs when dealing with complex, high-dimensional data.

Second, CNNs maintain the 2D spatial structure of images, preserving valuable contextual information such as spatial relationships between features. This is especially important in medical imaging, where the relative positions of anatomical structures can be crucial for accurate diagnosis. In contrast, SVMs flatten images into 1D vectors, effectively discarding the spatial relationships between pixels and losing critical context, which hinders their ability to perform well on image-based tasks like MRI classification.

Third, CNNs are more computationally efficient than SVMs because they use shared filters to detect patterns across the image. This weight-sharing mechanism reduces the number of parameters that need to be learned, making CNNs more scalable and efficient for large datasets. SVMs, on the other hand, treat each pixel as a separate feature, resulting in a large number of parameters that can be computationally expensive to process, especially as the dataset size increases. This leads to scalability issues for SVMs when dealing with large or high-dimensional datasets.

SVMs, on the other hand, are not well-suited for large and complex datasets, especially when the number of data points increases or the problem becomes more intricate. One major limitation of SVMs is the exponential increase in computation cost as more data points are added. Since SVMs calculate pairwise similarities (or kernel functions) between all training data points, this leads to a significant rise in the computational burden as the dataset grows. For large-scale problems like classifying knee MRIs, this exponential increase in computation time can become impractical, making SVMs inefficient and time-consuming compared to deep learning models like CNNs.

In addition, SVMs can struggle with heavily imbalanced datasets, where one class (e.g., healthy ACLs) is significantly larger than the other (e.g., damaged ACLs). In such cases, the SVM model can become biased toward the majority class, as the hyperplane tends to be skewed in favor of the class with more samples. This can result in high accuracy for the majority class but poor performance for the minority class, which is often the more clinically relevant class. While techniques like adjusting class weights can help mitigate this issue, SVMs may still struggle to achieve good performance, particularly when the class imbalance is extreme. In contrast, CNNs are more robust in handling such imbalances through techniques like data augmentation and learning hierarchical features that improve generalization to the minority class.

In summary, CNNs outperform SVMs for knee MRI classification because they automatically learn hierarchical features, maintain the spatial structure of images, and are computationally more efficient. SVMs, by contrast, require manual feature engineering, lose spatial context through image flattening, and struggle with large, complex datasets due to their high computational cost and poor handling of class imbalance. These limitations make SVMs less effective for tasks like MRI classification, where deep learning models like CNNs excel.

5 Dataset Challenges

- **Class Imbalance:** The dataset contained an unequal number of images for each class. This imbalance caused biased predictions, particularly for the underrepresented class. Techniques such as oversampling, undersampling, and weighted loss functions were explored to mitigate this issue.
- **Image Quality and Variability:** Some MRI scans in the dataset had inconsistencies in image resolution, orientation, and quality. These variations likely due to the difference in imaging equipment, scanning protocols, and setting when acquiring the data. Such inconsistencies are challenging for the machine learning models to analyze and makes it efficient. To address this we have used preprocessing steps like resizing, normalization, and reshaping, that were necessary to standardize the input data.
- **Data Volume:** The dataset was relatively small for training a CNN effectively. CNN would perform better on the larger datasets due to their hgih capacity for learning complex features.
- **Processing Time:** Training CNN requires significant computational resources and time. High-performance hardware such as GPUs would have make the computation much easier, but due to the limitations of personal laptops, it made the working with complex dataset like this technically difficult.

6 Conclusion

This study compared the performance of Support Vector Machines (SVM) and Convolutional Neural Networks (CNN) in classifying knee MRI scans. The results demonstrated that CNNs significantly outperformed SVMs due to their ability to learn hierarchical features and retain spatial information from images. While the SVM struggled with flattened image data and high dimensionality, the CNN leveraged convolutional layers to extract meaningful patterns, achieving an accuracy of 88.51%.

The main challenges included class imbalance, limited data volume, and variability in image quality. Data augmentation and preprocessing techniques helped address these issues to some extent, but further improvements could be made by increasing the dataset size and exploring advanced CNN architectures, such as transfer learning with pre-trained models.

Overall, this study highlights the potential of deep learning, particularly CNNs, in automating medical image analysis. Future work could focus on optimizing model performance and interpretability, as well as exploring ensemble methods to combine the strengths of SVMs and CNNs for better diagnostic accuracy.

7 References

https://www.geeksforgeeks.org/support-vector-machines-vs-neural-networks/

https://peyrone.medium.com/comparing-svm-and-cnn-in-recognizing-handwritten-digits-an-overview-5ef06b20194e